

Влияние мемоизации на производительность React-приложения

Гагулия Нугзар

@NookieGrey



Frontend
Conf 2022

Обо мне

- 10 лет коммерческого опыта, работал в Яндекс, Альфа-Банк
 - Спикер Google I/O, Google Dev Fest
 - Автор статей на Хабре
 - Контрибьютор, ментор
- t.me/NookieGrey



Frontend
Conf 2022



Содержание

- Исходники useMemo
- Бенчмарки
- Когда использовать
- Когда не использовать
- Влияние на Реальные проекты
- Принципы оптимизации

Исходники useMemo



```
function useMemo(create, deps) {  
  var dispatcher = resolveDispatcher();  
  return dispatcher.useMemo(create, deps);  
}  
  
function resolveDispatcher() {  
  var dispatcher = ReactCurrentDispatcher.current;  
  if (dispatcher === null) {  
    error('Hooks can only be called inside of the body of a function component.');  }  
  
  return dispatcher;  
}
```

Исходники useMemo



```
var ReactSharedInternals = {  
  ReactCurrentDispatcher: ReactCurrentDispatcher,  
  // ...  
};  
  
exports.__SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED = ReactSharedInternals;  
  
// ---> react-dom.js
```

Исходники useMemo



```
function renderWithHooks(current, ...) {  
  if (current !== null && current.memoizedState !== null) {  
    ReactCurrentDispatcher.current = HooksDispatcherOnUpdateInDEV;  
  
  } else {  
    ReactCurrentDispatcher.current = HooksDispatcherOnMountInDEV;  
  }  
  
}
```

Исходники useMemo

```
HooksDispatcherOnUpdateInDEV = {
  useCallback: function (callback, deps) {/**/},
  useEffect: function (create, deps) {/**/},
  useMemo: function (create, deps) {
    currentHookNameInDev = 'useMemo';
    updateHookTypesDev();
    var prevDispatcher = ReactCurrentDispatcher.current;
    ReactCurrentDispatcher.current = InvalidNestedHooksDispatcherOnUpdateInDEV;
    try {
      return updateMemo(create, deps);
    } finally {
      ReactCurrentDispatcher.current = prevDispatcher;
    }
  }
};
```

Исходники useMemo

```
HooksDispatcherOnUpdateInDEV = {  
  useCallback: function (callback, deps) {/**/},  
  useEffect: function (create, deps) {/**/},  
  useMemo: function (create, deps) {  
    currentHookNameInDev = 'useMemo';  
    updateHookTypesDev();  
    var prevDispatcher = ReactCurrentDispatcher.current;  
    ReactCurrentDispatcher.current = InvalidNestedHooksDispatcherOnUpdateInDEV;  
    try {  
      return updateMemo(create, deps);  
    } finally {  
      ReactCurrentDispatcher.current = prevDispatcher;  
    }  
  }  
};
```


Исходники useMemo

```
function useMemo(nextCreate, deps) {  
  var hook = updateWorkInProgressHook();  
  var nextDeps = deps === undefined ? null : deps;  
  var prevState = hook.memoizedState;  
  if (prevState !== null && nextDeps !== null) {  
    var prevDeps = prevState[1];  
    if (areHookInputsEqual(nextDeps, prevDeps)) {  
      return prevState[0];  
    }  
  }  
  var nextValue = nextCreate();  
  hook.memoizedState = [nextValue, nextDeps];  
  return nextValue;  
}
```

Исходники useMemo

```
function useMemo(nextCreate, deps) {  
  var hook = updateWorkInProgressHook();  
  var nextDeps = deps === undefined ? null : deps;  
  var prevState = hook.memoizedState;  
  if (prevState !== null && nextDeps !== null) {  
    var prevDeps = prevState[1];  
    if (areHookInputsEqual(nextDeps, prevDeps)) {  
      return prevState[0];  
    }  
  }  
  var nextValue = nextCreate();  
  hook.memoizedState = [nextValue, nextDeps];  
  return nextValue;  
}
```



Исходники useMemo

```
function updateMemo(nextCreate, deps) {  
  var hook = updateWorkInProgressHook();  
  var nextDeps = deps === undefined ? null : deps;  
  var prevState = hook.memoizedState;  
  if (prevState !== null && nextDeps !== null) {  
    var prevDeps = prevState[1];  
    if (areHookInputsEqual(nextDeps, prevDeps)) {  
      return prevState[0];  
    }  
  }  
  var nextValue = nextCreate();  
  hook.memoizedState = [nextValue, nextDeps];  
  return nextValue;  
}
```



Исходники useMemo

```
function updateWorkInProgressHook() {
  var newHook = {
    memoizedState: currentHook.memoizedState,
    baseState: currentHook.baseState,
    baseQueue: currentHook.baseQueue,
    queue: currentHook.queue,
    next: null
  };
  if (workInProgressHook === null) {
    currentlyRenderingFiber$1.memoizedState = workInProgressHook = newHook;
  } else {
    workInProgressHook = workInProgressHook.next = newHook;
  }

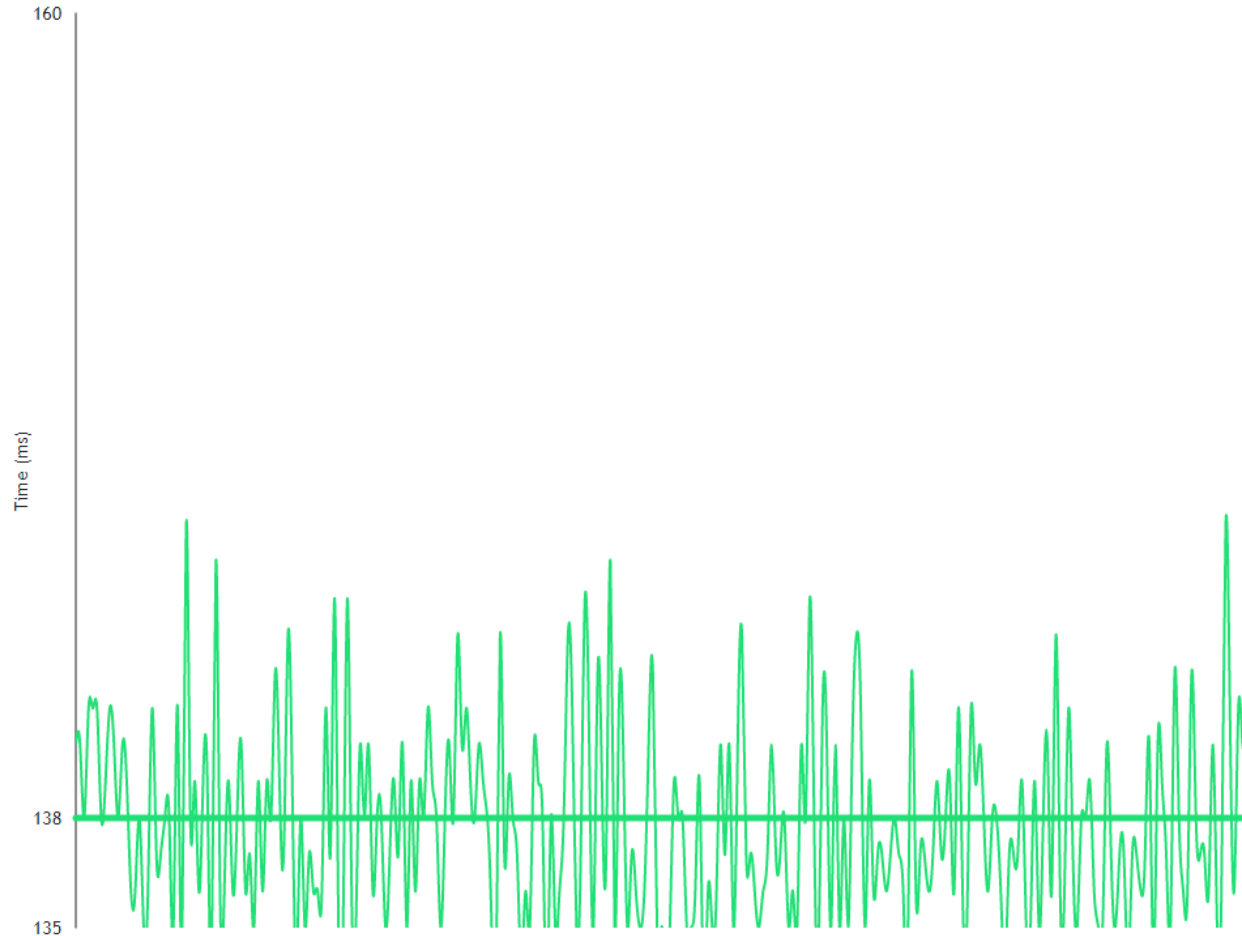
  return workInProgressHook;
}
```

Исходники useMemo

```
function useMemo(nextCreate, deps) {  
  var hook = updateWorkInProgressHook();  
  var nextDeps = deps === undefined ? null : deps;  
  var prevState = hook.memoizedState;  
  if (prevState !== null && nextDeps !== null) {  
    var prevDeps = prevState[1];  
    if (areHookInputsEqual(nextDeps, prevDeps)) {  
      return prevState[0];  
    }  
  }  
  var nextValue = nextCreate();  
  hook.memoizedState = [nextValue, nextDeps];  
  return nextValue;  
}
```

Издержки мемоизации

Мемоизация увеличивает TTI рендера на 10-15%

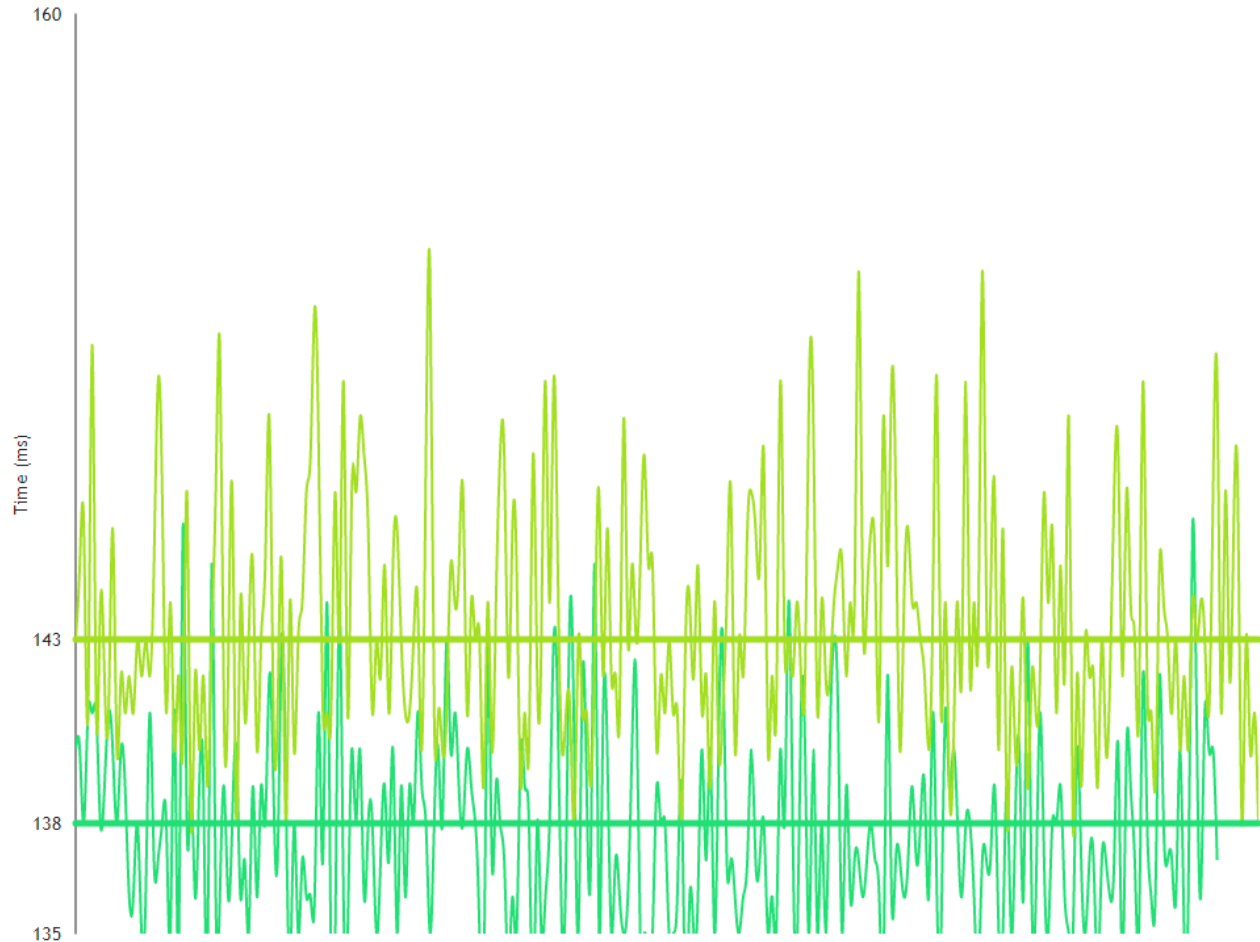


<input type="checkbox"/>	<input type="checkbox"/>	memo	useMemo	useCallback	<input type="checkbox"/> Axis
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	---	---	<input checked="" type="checkbox"/> 138
<input type="checkbox"/>	<input type="checkbox"/>	---	---	true	<input type="checkbox"/> 141
<input type="checkbox"/>	<input type="checkbox"/>	---	true	---	<input type="checkbox"/> 142
<input type="checkbox"/>	<input type="checkbox"/>	---	true	true	<input type="checkbox"/> 143
<input type="checkbox"/>	<input type="checkbox"/>	true	---	---	<input type="checkbox"/> 145
<input type="checkbox"/>	<input type="checkbox"/>	true	---	true	<input type="checkbox"/> 149
<input type="checkbox"/>	<input type="checkbox"/>	true	true	---	<input type="checkbox"/> 149
<input type="checkbox"/>	<input type="checkbox"/>	true	true	true	<input type="checkbox"/> 150

*TTI – time to interaction

Издержки мемоизации

Мемоизация увеличивает TTI рендера на 10-15%

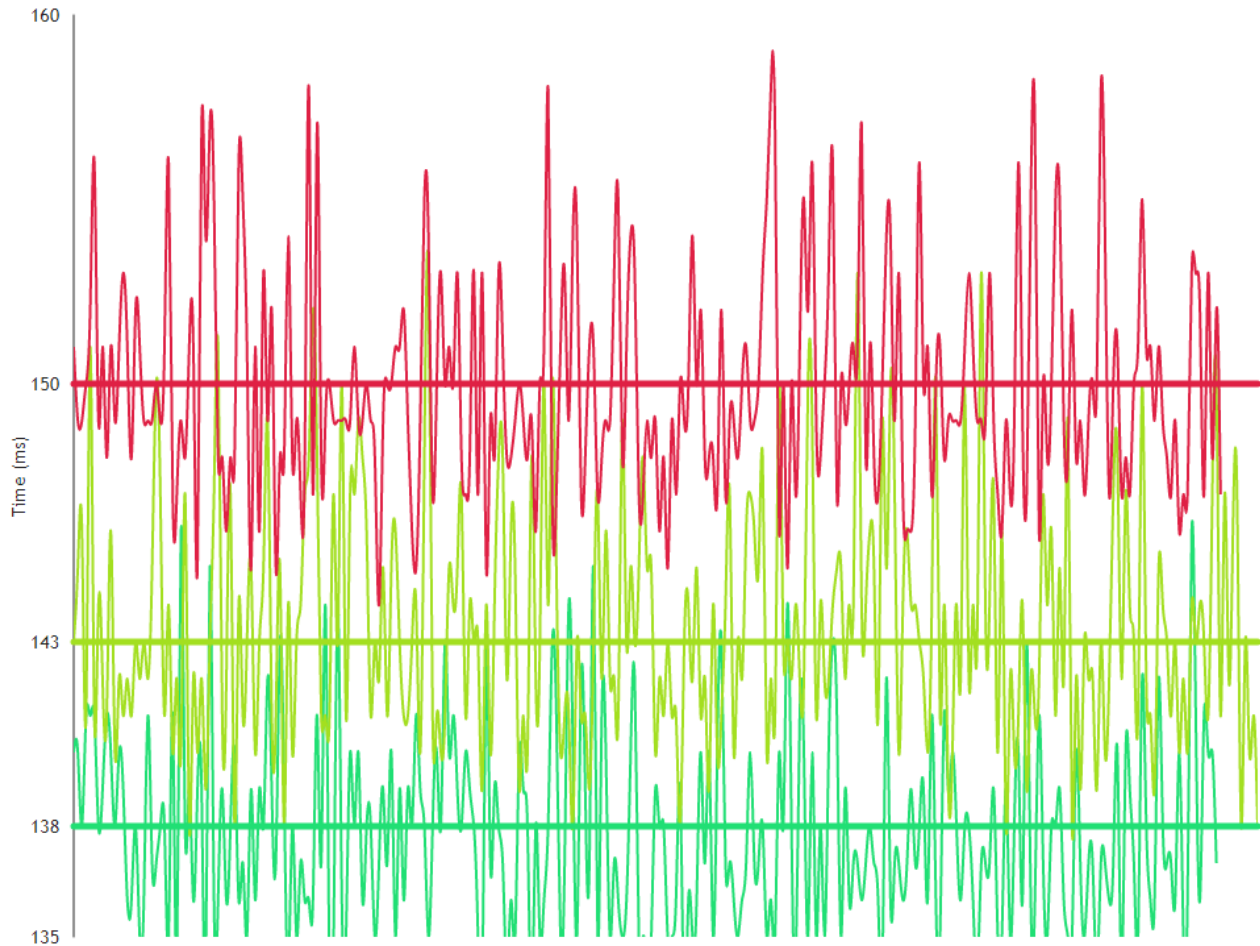


<input type="checkbox"/>	<input type="checkbox"/>	memo	useMemo	useCallback	<input type="checkbox"/> Axis
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	---	---	<input checked="" type="checkbox"/> 138
<input type="checkbox"/>	<input type="checkbox"/>	---	---	true	<input type="checkbox"/> 141
<input type="checkbox"/>	<input type="checkbox"/>	---	true	---	<input type="checkbox"/> 142
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	true	true	<input checked="" type="checkbox"/> 143
<input type="checkbox"/>	<input type="checkbox"/>	true	---	---	<input type="checkbox"/> 145
<input type="checkbox"/>	<input type="checkbox"/>	true	---	true	<input type="checkbox"/> 149
<input type="checkbox"/>	<input type="checkbox"/>	true	true	---	<input type="checkbox"/> 149
<input type="checkbox"/>	<input type="checkbox"/>	true	true	true	<input type="checkbox"/> 150

*TTI – time to interaction

Издержки мемоизации

Мемоизация увеличивает TTI рендера на 10-15%



<input type="checkbox"/>	<input type="checkbox"/>	memo	useMemo	useCallback	<input type="checkbox"/> Axis
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	---	---	<input checked="" type="checkbox"/> 138
<input type="checkbox"/>	<input type="checkbox"/>	---	---	true	<input type="checkbox"/> 141
<input type="checkbox"/>	<input type="checkbox"/>	---	true	---	<input type="checkbox"/> 142
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	true	true	<input checked="" type="checkbox"/> 143
<input type="checkbox"/>	<input type="checkbox"/>	true	---	---	<input type="checkbox"/> 145
<input type="checkbox"/>	<input type="checkbox"/>	true	---	true	<input type="checkbox"/> 149
<input type="checkbox"/>	<input type="checkbox"/>	true	true	---	<input type="checkbox"/> 149
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	true	true	true	<input checked="" type="checkbox"/> 150

*TTI – time to interaction

Когда использовать React-мемоизацию

- В зависимостях `useEffect`
- Для супердорогих вычислений
- Для высоконагруженных компонентов

Использование в зависимостях useEffect()

```
import {createContext, useCallback, useEffect} from "react";

const Context = createContext({});

export function NotifyContext({children}) {
  const notify = useCallback(() => {
    //...
  }, []);
  return (
    <Context.Provider value={{notify}}>
      {children}
    </Context.Provider>
  )
}
```

Использование в зависимостях useEffect()

```
import {createContext, useCallback, useEffect} from "react";

const Context = createContext({});


export function NotifyContext({children}) {
  const notify = useCallback(() => {
    //...
  }, []);
  return (
    <Context.Provider value={{notify}}>
      {children}
    </Context.Provider>
  )
}
```

```
import {useContext} from "react";

export function Component() {
  const {notify} = useContext(Context);

  useEffect(() => {
    fetch()
      .then(() => {
        //...
      })
      .catch(error => {
        notify(error.message)
      })
  }, [notify])
}
```

Сложные вычисления



```
export function Component(count) {  
  const fibonacci = useMemo(() => {  
    calculate(count);  
  }, [count]);  
  
  return (  
    <>  
      { /* ... */ }  
    </>  
  )  
}
```

Высоконагруженный компонент



```
const VeryExpensiveTree = memo(VeryExpensiveTreeComponent);

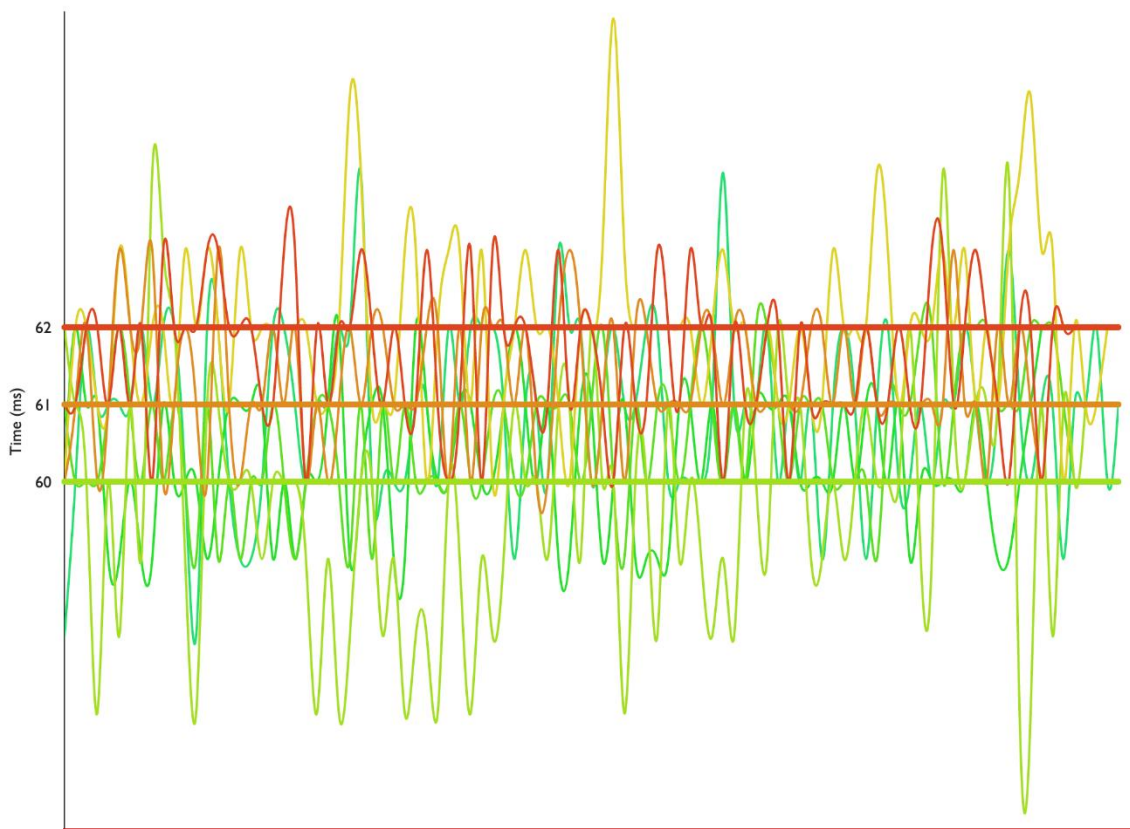
export function Page() {
  const [state, setState] = useState(0);

  const handler = useCallback((event) => {
    setState(event.target.value)
  }, [])

  return (
    <Wrapper>
      <OtherComponent state={state}/>
      <VeryExpensiveTree handler={handler}/>
    </Wrapper>
  )
}
```

Триплет

Эффект только при полной мемоизации



<input type="checkbox"/>	<input type="checkbox"/>	memo	useMemo	useCallback	<input type="checkbox"/> Axis
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	---	---	<input checked="" type="checkbox"/> 61
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	---	true	<input checked="" type="checkbox"/> 60
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	true	---	<input checked="" type="checkbox"/> 61
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	---	true	true	<input checked="" type="checkbox"/> 60
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	true	---	---	<input checked="" type="checkbox"/> 62
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	true	---	true	<input checked="" type="checkbox"/> 61
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	true	true	---	<input checked="" type="checkbox"/> 62
<input type="checkbox"/>	<input type="checkbox"/>	true	true	true	<input type="checkbox"/> 0

*Триплет – полный набор из 3 элементов

Когда НЕ использовать React-мемоизацию

- Для предотвращения избыточного рендера
- Полное покрытие проекта мемоизацией

← Твит



ДЭН
@dan_abramov



Why doesn't React put memo() around every component by default? Isn't it faster? Should we make a benchmark to check?

Ask yourself:

Why don't you put Lodash memoize() around every function? Wouldn't that make all functions faster? Do we need a benchmark for this? Why not?

5:22 AM · 12 янв. 2019 г. · Twitter Web App

44 ретвита 8 твитов с цитатами 321 отметка «Нравится»

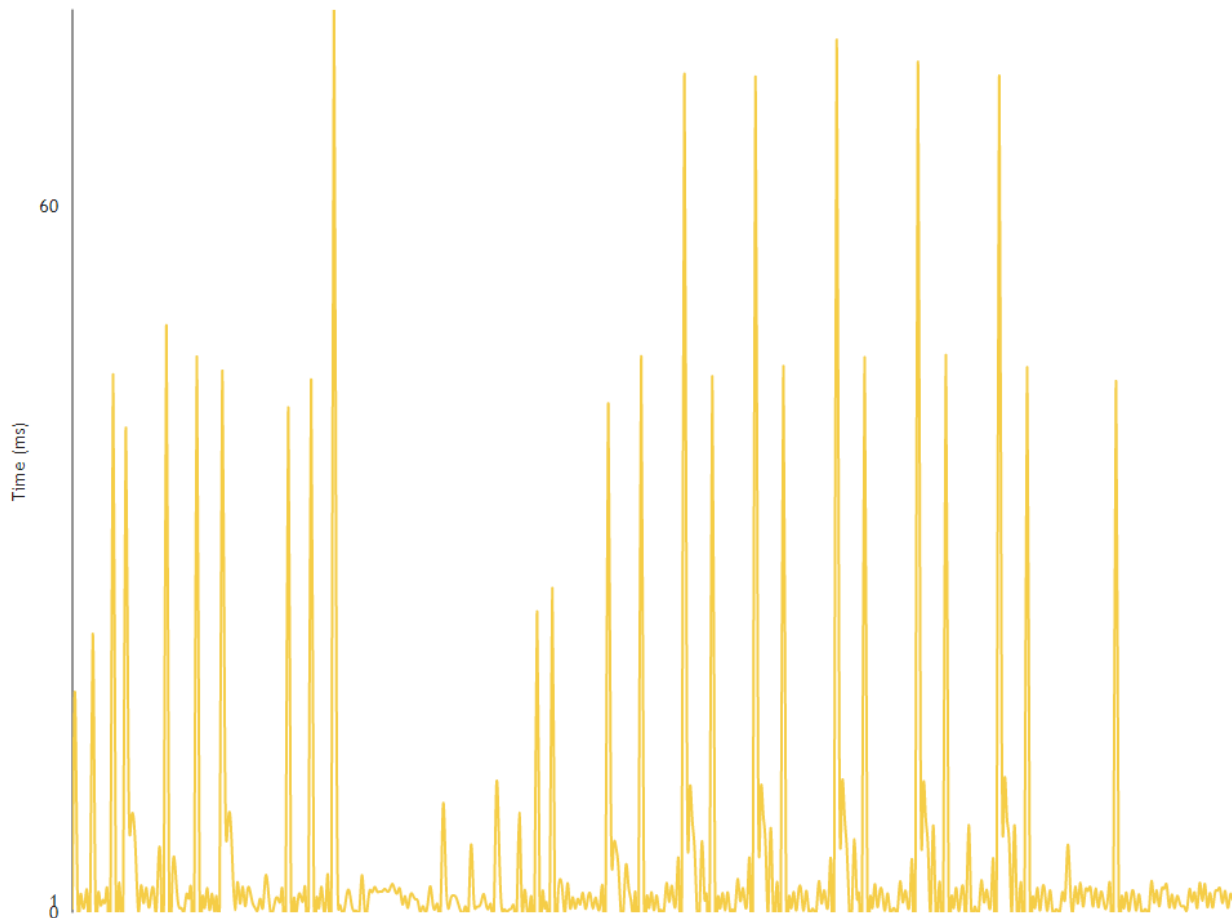


Избыточный рендер

```
export function Page() {  
  const [state, setState] = useState(0);  
  const handler = (event) => {  
    setState(event.target.value)  
  }  
  
  return (  
    <Form>  
      <Input value={state} onChange={handler}/>  
      { /* ... */ }  
    </Form>  
  )  
}
```


Реальный проект

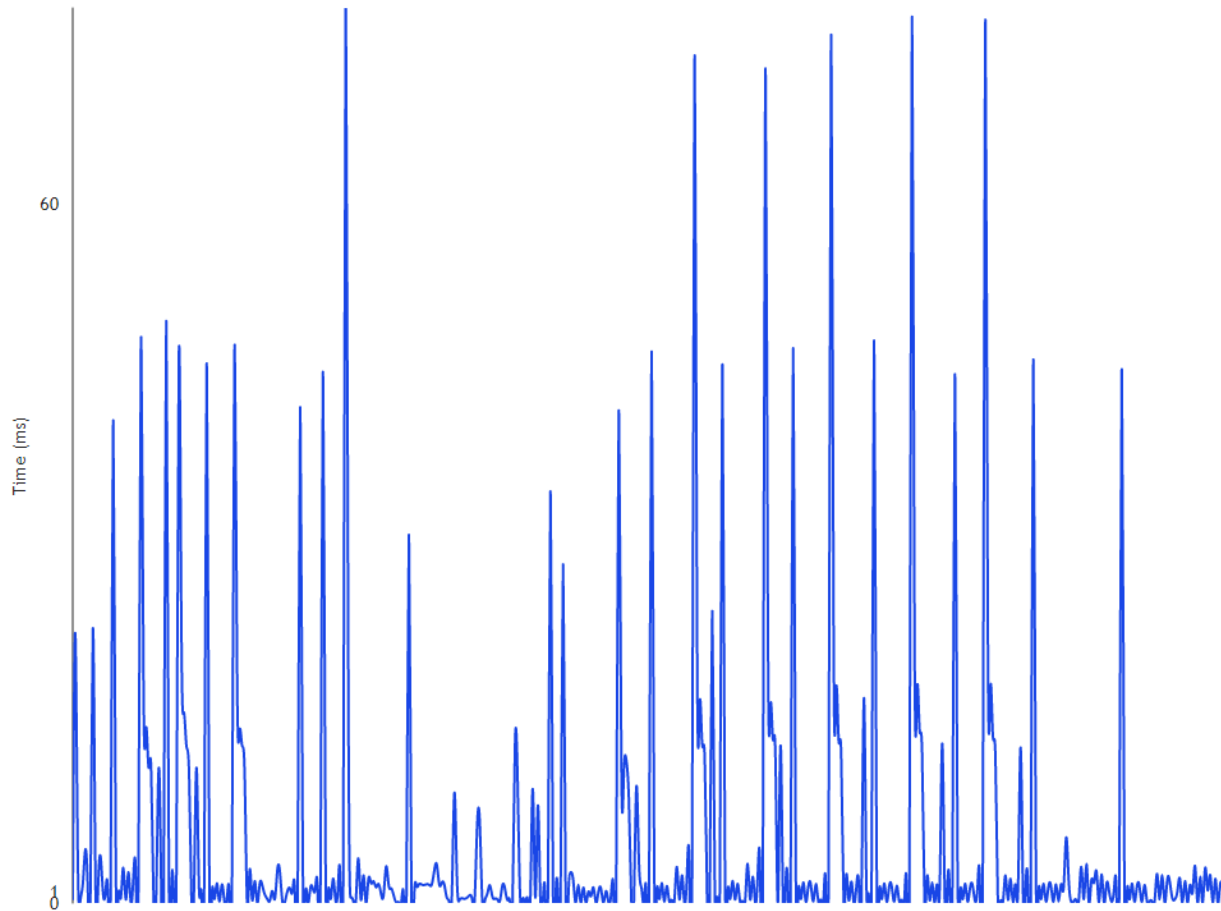
Полностью покрытый мемоизацией



<input type="checkbox"/>	<input type="checkbox"/>	Name	<input type="checkbox"/>	Axis
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	memoized	<input checked="" type="checkbox"/>	1
<input type="checkbox"/>	<input type="checkbox"/>	without	<input type="checkbox"/>	1

Реальный проект

Переопределяем мемо

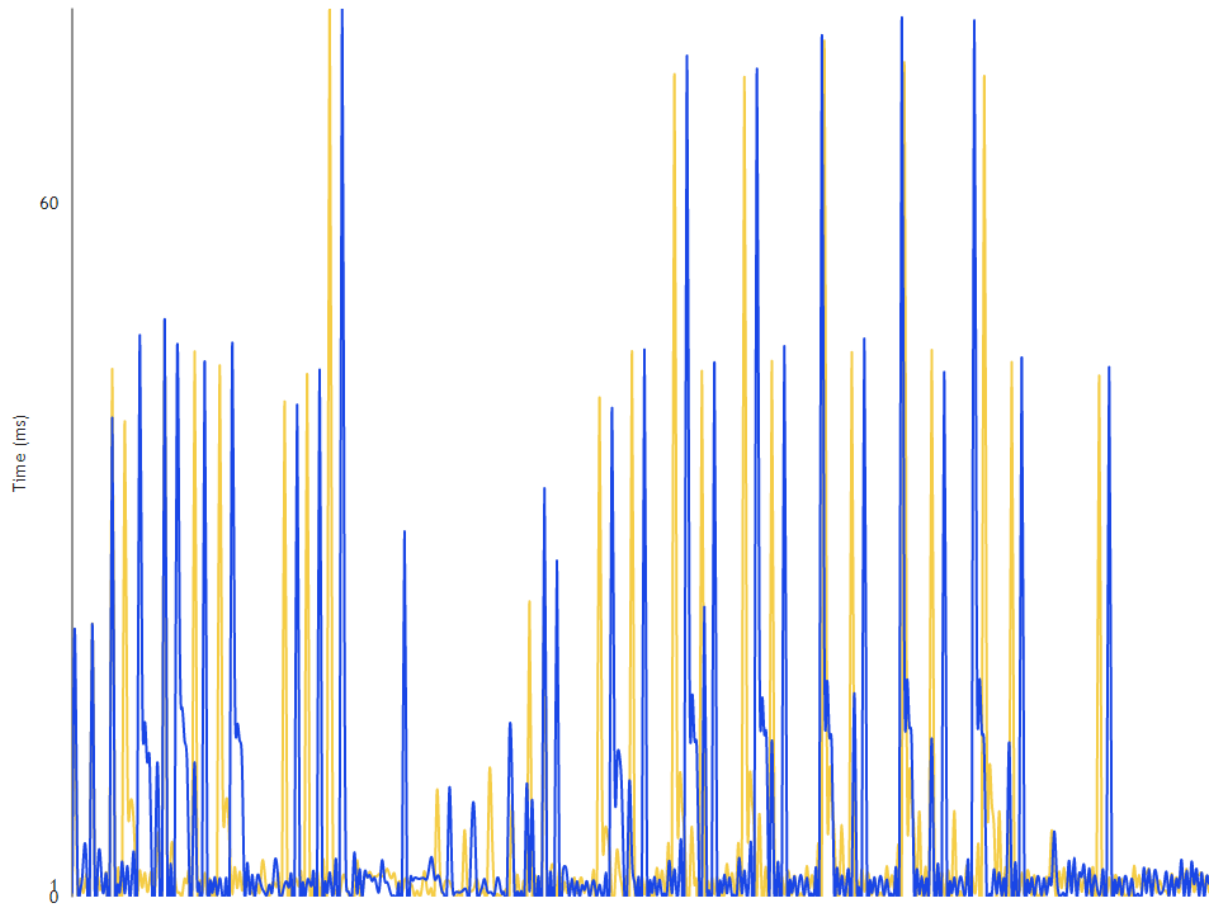


<input type="checkbox"/>	<input type="checkbox"/>	Name	<input type="checkbox"/>	Axis
<input type="checkbox"/>	<input type="checkbox"/>	memoized	<input type="checkbox"/>	1
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	without	<input checked="" type="checkbox"/>	1

```
React.memo = (Component) => {  
  return Component;  
};
```

Реальный проект

Переопределяем мемо

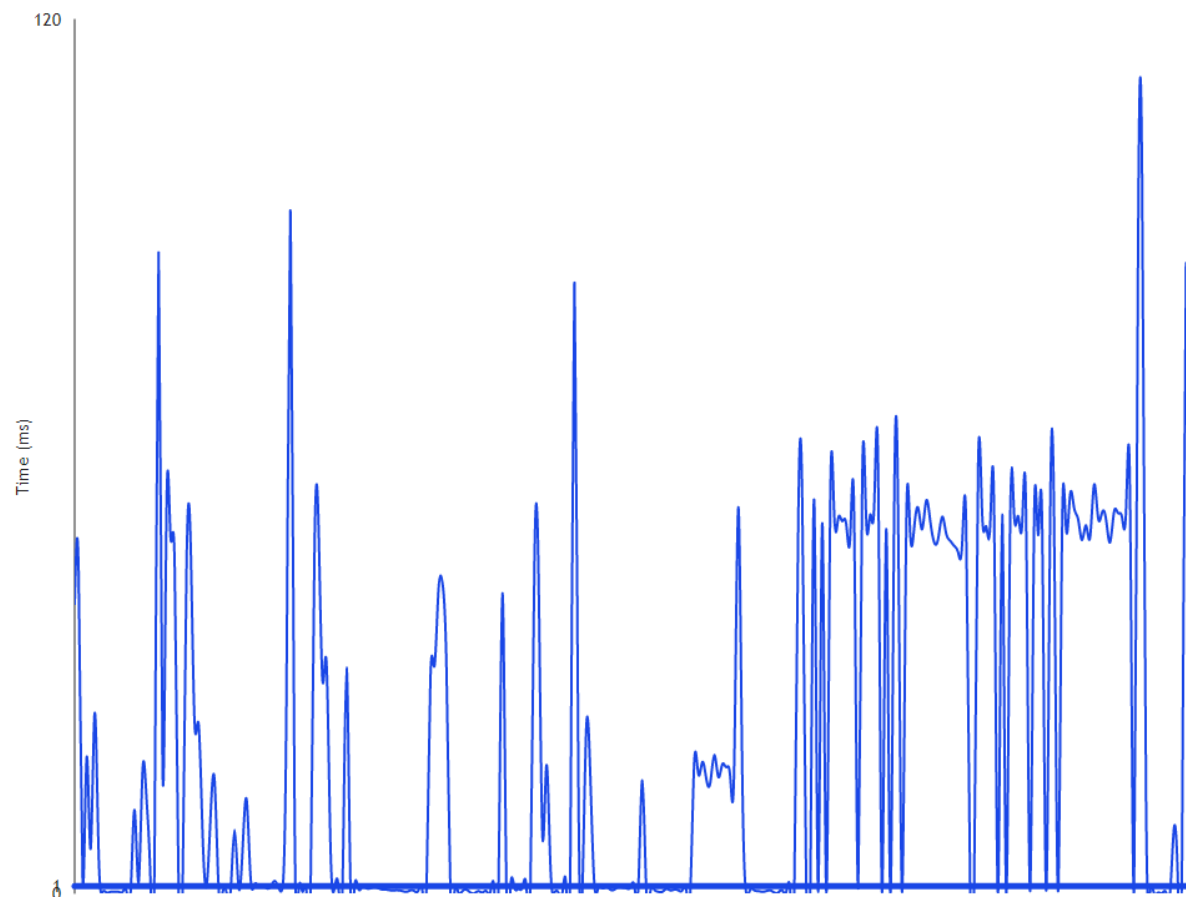


✓	✓	Name	✓	Axis
✓	✓	memoized	✓	1
✓	✓	without	✓	1

```
React.memo = (Component) => {  
  return Component;  
};
```

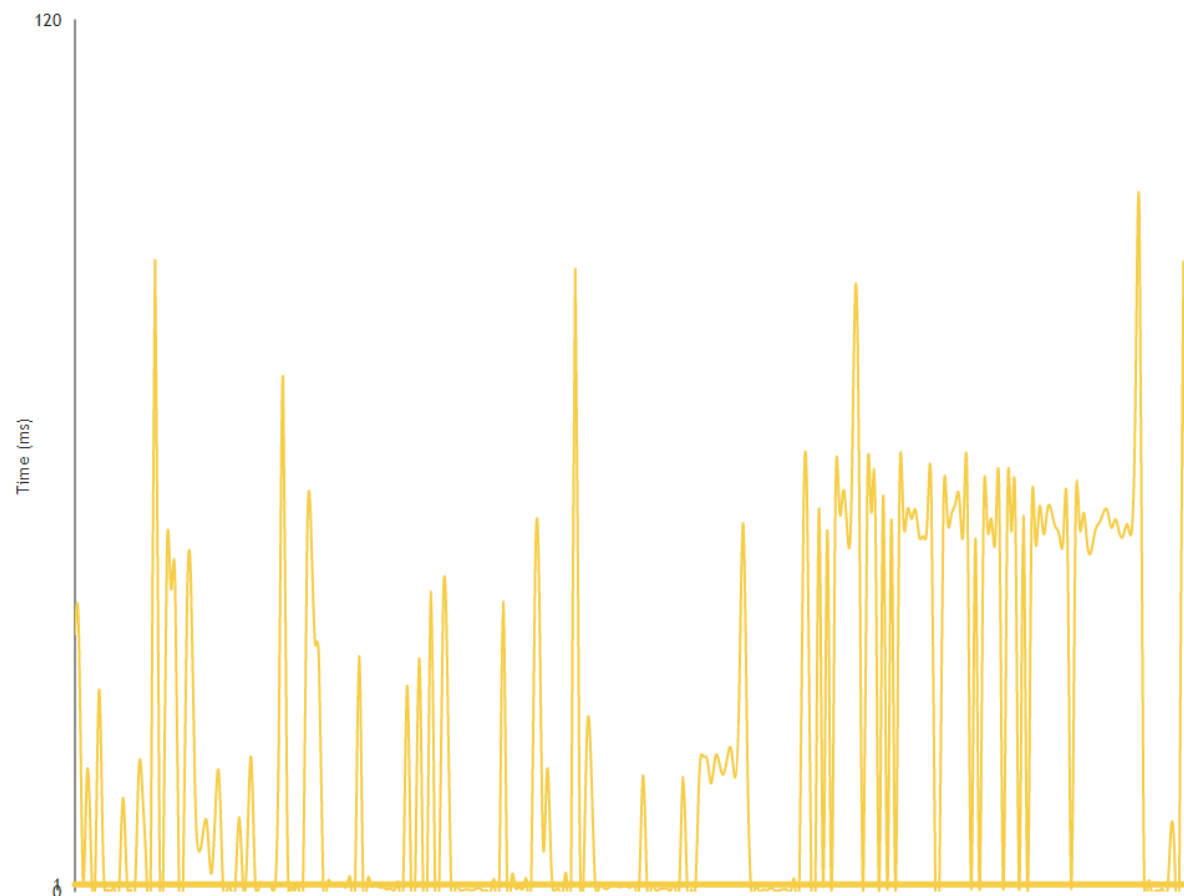
Реальный проект 2

Запускаем без мемо



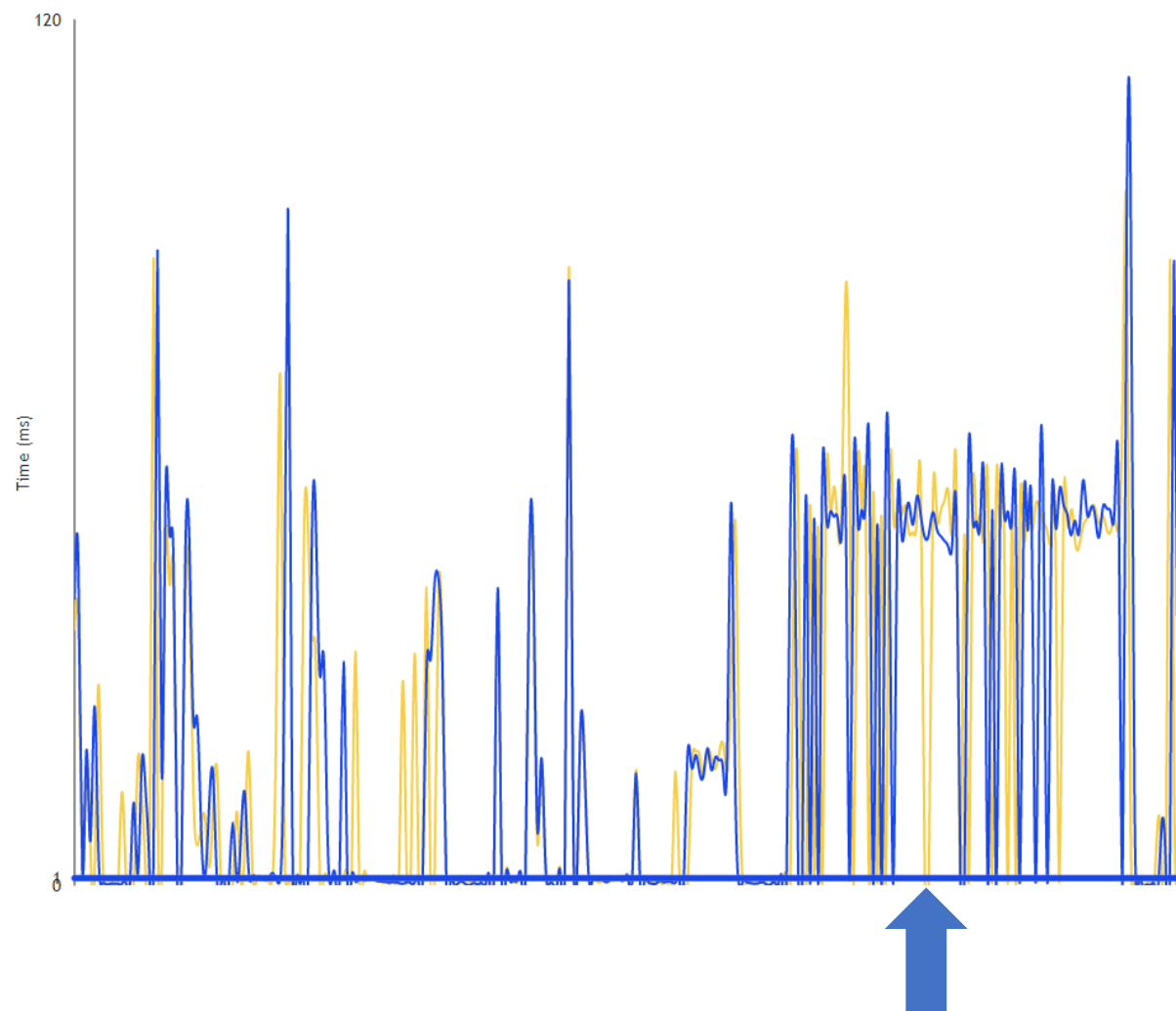
Реальный проект 2

Оборачиваем всё в мемо



Реальный проект 2

Оборачиваем всё в мемо



Чек-лист

- › Принцип Целесообразности
- › Преждевременная оптимизация
- › Необоснованное усложнение кода
- › Оптимизация узких мест
- › Оптимизация ради оптимизации



Прежде чем начать оптимизировать

Принцип Целесообразности

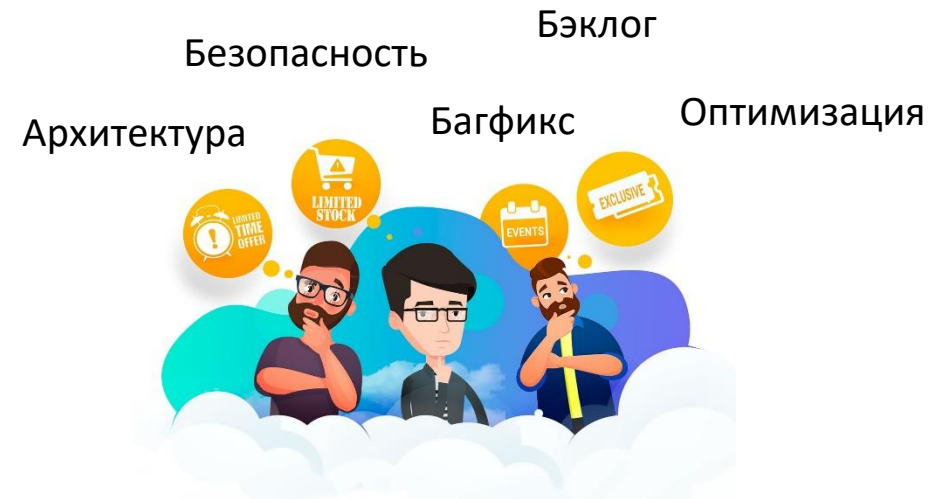
- | Есть ли потребность в оптимизации?
- › Преждевременная оптимизация
- › Необоснованное усложнение кода
- › Оптимизация узких мест
- › Оптимизация ради оптимизации



Цель -> Средство -> Результат?!

Преждевременная оптимизация

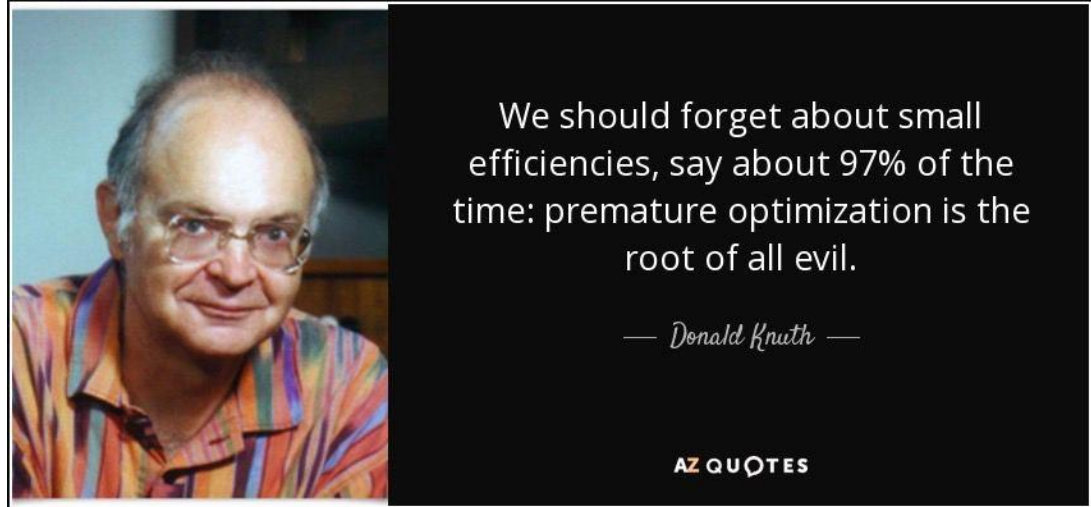
- › Принцип Целесообразности
- | Приоритетна ли оптимизация?
- › Необоснованное усложнение кода
- › Оптимизация узких мест
- › Оптимизация ради оптимизации



безграничные потребности VS ограниченные ресурсы

Необоснованное усложнение кода

- › Принцип Целесообразности
- › Преждевременная оптимизация
- | Оправдана ли оптимизация?
- › Оптимизация узких мест
- › Оптимизация ради оптимизации



Управление сложностью

Оптимизация узких мест

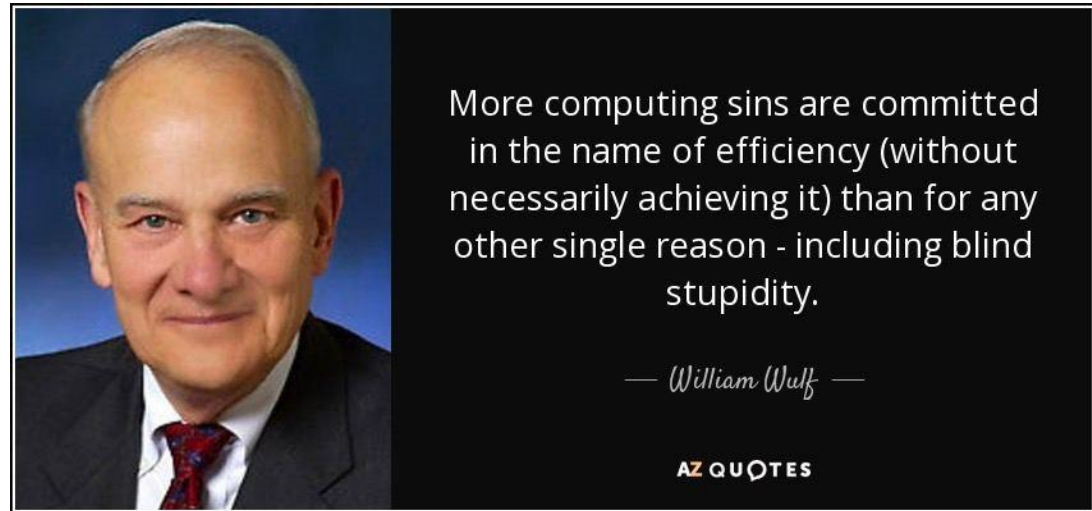
- › Принцип Целесообразности
- › Преждевременная оптимизация
- › Необоснованное усложнение кода
- | Верные ли участки кода мы оптимизируем?
- › Оптимизация ради оптимизации



Бутылочное горлышко

Оптимизация ради оптимизации

- › Принцип Целесообразности
 - › Преждевременная оптимизация
 - › Необоснованное усложнение кода
 - › Оптимизация узких мест
- | Не обманываем ли мы себя?



Необдуманная оптимизация приносит вред

Мемоизация

- Недешёвая
- Работает только в триплете
- Используется в 99% в `useEffect`
- Есть ли потребность?
- Есть ли ресурсы?
- Больше выгоды, чем вреда?
- Заточена под бутылочное горлышко?
- Приносит ли профит?

Секция вопросов

Не используйте
повсеместную мемоизацию
только для предотвращения
избыточного рендера

Гагулия Нугзар
@NookieGrey



FC

Frontend
Conf 2022